Activating a research fault localization bot on your project? #595

Edit New issue

Open dginelli opened this issue 26 days ago · 6 comments

dginelli commented 26 days ago — Spammy

Hi!

We are software engineering researchers and are currently looking for active open-source projects where we could test a new cool software bot: flaccocobot!

When a new pull request is open, if it has test failures/errors, flaccocobot analyzes the failure and posts a diagnosis as comments to the pull request, highlighting the most suspicious lines that likely need to be analyzed to fix the bug.

flaccocobot works as follows:

- When a new PR is open, if the test suite fails, flaccocobot starts its analysis.
- The top k suspicious (potentially buggy) lines of code are then highlighted as comments to the PR.

We would love to have your feedback in order to understand what you like or you do not like, what we can improve, and if you find it useful!

If you are interested in trying it, we will run flaccocobot on your project for a period of two weeks and then give you a 1-minute questionnaire to collect your feedback.

What do you think? Can we activate flaccocobot for classgraph?

Thanks,

Davide (**@dginelli**), André (**@andre15silva**), Matias (**@martinezmatias**), Benjamin (**@danglotb**), Martin (**@monperrus**)

You can find out more in our tools repos:

- https://github.com/SpoonLabs/flacoco
- https://github.com/eclipse/repairnator

Assignees: No one assigned
Labels: None yet
Projects: None yet
Milestone: No milestone
Linked pull requests: Successfully merging a pull request may close this issue. None yet
Notifications: Customize — Unsubscribe — You're receiving notifications because you were mentioned.
2 participants

lukehutch commented 26 days ago — Member

Sure, why not! This sounds pretty cool, and I already use a few other static analysis bots.

So I'm guessing for flacoco, you do automatic git-bisect to find the commit that broke one or more tests, then you do dataflow analysis to find the lines in that commit that are in a path to a broken test?

As far as repairnator, I will say that the idea of automated program repair makes me feel very nervous... how good are the suggestions in general? (I see the stat that five out of twelve fixes were accepted in one trial.)

dginelli commented 25 days ago — Author — Spammy

Hello **@lukehutch**,

> Sure, why not! This sounds pretty cool, and I already use a few other static analysis bots.

Great, we are happy to see that you would like to test flaccocobot and that you are interested about its functioning.

I give you more details to better explain how it works.

> So I'm guessing for flacoco, you do automatic git-bisect to find the commit that broke one or more tests, then you do dataflow analysis to find the lines in that commit that are in a path to a broken test?

When flaccocobot starts to scan a project, it searches for the open and failed pull requests created in the last 7 days. Then, for the next iterations, it searches for new pull requests or updates in the pull requests previously inspected (without considering updates related to the addition of new comments or review comments).

When flaccocobot finds a pull request that satisfies the requirements, it clones the source code associated with that pull request in an isolated Docker environment and it executes the test cases associated with the project.

Flacoco localization tool uses JaCoCo to instrument the program under analysis and record the covered lines by each test case. This step is important to allow Flacoco to associate a suspiciousness score to every line. For example, if a line is executed only by test cases that pass, this line has a suspiciousness score lower than a line that is executed only by test cases that end with a failure or an error.

After Flacoco localization finds all the lines that might be associated with the bug, the last step is to check which lines are in the diff introduced by the pull request compared to the original source code. flaccocobot suggests the top 5 most suspicious lines, contained also in the diff, in a review comment to the analyzed pull request.

> As far as repairnator, I will say that the idea of automated program repair makes me feel very nervous... how good are the suggestions in general? (I see the stat that five out of twelve fixes were accepted in one trial.)

About program repair, there are many different strategies and it is an active and challenging research field. To have an idea, you can see here how many different strategies and studies are conducted in this field.

The initial approaches were based on genetic programming to mutate a program by deleting/adding/changing a piece of code in the program under analysis with other pieces of code contained in the same program. This was based on the plastic surgery hypothesis that says: "Changes to a codebase contain snippets that already exist in the codebase at the time of the change, and these snippets can be efficiently found and exploited".

Now there are also approaches that exploit machine/deep learning in order to create better patches for bugs by learning from what developers did in the past.

One of the main problem about generating patches is the so called "overfitting patches", that are patches that make the program pass all the test cases, but they are actually incorrect. This means for example that the tests do not cover every aspect of the program, and so a patch that makes the program pass all the test cases may add a side effect not revealed by any test case.

Program repair tools have a phase dedicated to the localization of the points in a program that are more likely to be changed to create a patch. Flacoco is a new fault localization tool that can be exploited by program repair techniques and also by our new bot (flaccocobot) that we are currently testing.

I can say that we do not have a perfect system right now to create patches that are 100% correct, especially when the bug requires a lot of changes, but we are trying to do our best to improve the systems :-)

lukehutch commented 24 days ago — Member

@dginelli Thank you for the incredibly detailed, interesting, and helpful information!

That list of program repair strategies is impressive.

> One of the main problem about generating patches is the so called "overfitting patches", that are patches that make the program pass all the test cases, but they are actually incorrect.

This is one of my greatest fears with automated program repair. The other greatest fear is that a program repair tool will suggest tools that will trick a human into accepting them, because they look so plausible on the surface, while being subtly wrong. The human tendency towards confirmation bias will often blind the user into not being able to cognitively process this subtlety.

This is why GitHub Copilot is probably more dangerous than it is helpful -- lots of examples of how this can go wrong were posted on the Web after this tool was released. Copilot is basically a tool for *amplifying errors!* See if you can spot the error here that is shockingly wrong, but looks completely plausible on the surface:

```
const seconds = 1000
const minutes = 60 * seconds
const hours = 60 * minutes
const days = 24 * hours
const weeks = 7 * days
const months = 30 * days
const years = 12 * months
```

I'm glad people are working on automated program analysis, especially since I am a huge fan of static analysis. But also the computer scientist in me likes remember that many of the substantive problems in program analysis are uncomputable in the general case. It doesn't mean it's not worth pursuing, but it's an infinitely long tail to chase.

Anyway though, keep up the good work! And I definitely want to try this bot. Let me know how to set it up.

One thing I should point out though is that ClassGraph is extremely stable now, and has a very low defect rate (I fix about one bug every 3 months at this point, and almost all of those are very minor, niche, or subtle, i.e. would not break any tests). I even used to disallow git push operations if the tests didn't pass, via checkin hooks (although that is switched off right now). So I don't know if ClassGraph is the best project to test your bot with, because the tests won't fail often.

dginelli commented 24 days ago — Author — Spammy

Hello **@lukehutch**,

I'm glad to see that you appreciated the information and it is really nice to see your interest about these topics. It's a very stimulating conversation, thank you! :-)

> This is one of my greatest fears with automated program repair. The other greatest fear is that a program repair tool will suggest tools that will trick a human into accepting them, because they look so plausible on the surface, while being subtly wrong. The human tendency towards confirmation bias will often blind the user into not being able to cognitively process this subtlety.

This is a very interesting and crucial point! Based on some studies conducted in the past, it seems that developers tend not to trust very much patches generated by automated program repair tools. They find patches generated by developers more trustworthy than the ones automatically generated.

This does not always happen (e.g., we saw that some patches proposed by Repairnator have been accepted in the past) and other studies show that developers tend to trust more automatically generated patches if these have been generated using human strategies (i.e., patches generated exploiting machine/deep learning approaches that allow program repair tools to learn how to create patches studying the ones already generated by the developers in the past).

However, as you said, there is the risk about the acceptance of suggestions/patches by humans without putting the right level of attention, thus potentially creating undesired effects. And I think this is another interesting topic to be investigated.

> This is why GitHub Copilot is probably more dangerous than it is helpful -- lots of examples of how this can go wrong were posted on the Web after this tool was released. Copilot is basically a tool for *amplifying errors!* See if you can spot the error here that is shockingly wrong, but looks completely plausible on the surface:
>
> ```
> const seconds = 1000
> const minutes = 60 * seconds
> const hours = 60 * minutes
> const days = 24 * hours
> const weeks = 7 * days
> const months = 30 * days
> const years = 12 * months
> ```
>
> At first glance, as you said, it seems ok. Then, looking at it more carefully, the conversion of milliseconds in years does not work because it is true that a year has 12 months, but not every month has 30 days. So basically, using this code, you lose some days per year in the conversion.
>
> I'm glad people are working on automated program analysis, especially since I am a huge fan of static analysis. But also the computer scientist in me likes remember that many of the substantive problems in program analysis are uncomputable in the general case. It doesn't mean it's not worth pursuing, but it's an infinitely long tail to chase.
>
> Anyway though, keep up the good work! And I definitely want to try this bot. Let me know how to set it up.

Ok, great! :-) About activating the bot, we will now add your project to the list of the ones that flaccocobot can scan. You don't have to do anything else. After 14 days, flaccocobot will stop to scan the project (or even before if you change your idea), and then we will just ask you to fill a short questionnaire about the suggestions of flaccocobot (if it will have added some) and software bots in general.

> One thing I should point out though is that ClassGraph is extremely stable now, and has a very low defect rate (I fix about one bug every 3 months at this point, and almost all of those are very minor, niche, or subtle, i.e. would not break any tests). I even used to disallow git push operations if the tests didn't pass, via checkin hooks (although that is switched off right now). So I don't know if ClassGraph is the best project to test your bot with, because the tests won't fail often.

Ok, I see. Thank you for the information. If we will not be lucky and flaccocobot will not find any pull request to analyze in these 14 days, we will just ask you to fill a super short questionnaire about software bots in general.

lukehutch commented 24 days ago — Member

> This is a very interesting and crucial point! Based on some studies conducted in the past, it seems that developers tend not to trust very much patches generated by automated program repair tools. They find patches generated by developers more trustworthy than the ones automatically generated.
>
> This does not always happen (e.g., we saw that some patches proposed by Repairnator have been accepted in the past) and other studies show that developers tend to trust more automatically generated patches if these have been generated using human strategies

It's great to know that this has been studied!

> ```
> const seconds = 1000
> const minutes = 60 * seconds
> const hours = 60 * minutes
> const days = 24 * hours
> const weeks = 7 * days
> const months = 30 * days
> const years = 12 * months
> ```
>
> At first glance, as you said, it seems ok. Then, looking at it more carefully, the conversion of milliseconds in years does not work because it is true that a year has 12 months, but not every month has 30 days. So basically, using this code, you lose some days per year in the conversion.

That's a good guess, but no, that's not the actual problem with this code... think hard about it, it's a very good exercise... and I don't want to tell you what the answer is, because I don't want to deny you the eureka moment that you will get once you figure it out. Hint: it's a problem of semantics. (If you can't guess it, I'll tell you, but try again first...)

And that's really the problem with code recommendation in general using machine learning. Anything to do with semantics requires actual understanding to reason about it, and for understanding, we don't need just machine learning, we need AGI (Artificial General Intelligence). Nobody has any clue how to build AGI yet. So we are limited to only being able to make statistically likely guesses as to the correct fix, we can't develop tools that understand the problem and apply the correct fix without AGI.

dginelli commented 17 days ago — Author — Spammy

Hi **@lukehutch**,

so, since you mentioned the semantics, the first thing that I noticed when I looked at that piece of code was the fact the variables names are in the plural form, but actually they represent one single unit.

For example, one second has 1000ms, so the variables should be called `second`. One minute has 60 seconds, so I would have preferred to have the variable declared as `const minute = 60 * second;`.

Let me know if this is the correct answer or not :-)

Write Preview

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

Close issue Comment

Remember, contributions to this repository should follow our GitHub Community Guidelines.

Show your support for lukehutch by sponsoring them. Sponsor